(2)

Supercomputing Power for Interactive Visualization

This research is supported in part by
the Defense Advanced Research Projects Agency, DARPA ISTO Order No. 7510,
the National Science Foundation, Grant No. MIP-9000894,
and the Office of Naval Research, Contract No. N0014-86-K-0680

Report of Research Progress
October 1990 – March 1991

Henry Fuchs, Principal Investigator
John Poulton, Principal Investigator
John Eyles, Laura Weaver, Research Engineers
Mike Bajura, Andrew Bell, Jeff Butterworth, David Ellsworth, Howard Good, Edward Hill*,
Victoria Interrante, Jonathan Leech, Carl Mueller, Ulrich Neumann, Marc Olano, Mark Parris,
John Rhoades, Steve Molnar, Andre State, Greg Turk, Research Assistants

Department of Computer Science
Sitterson Hall
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175

* Hill is a part-time, non-student staff member.

1. SUMMARY

Our work over the past several years has explored multicomputer architectures for 3D graphics that offer the promise of scalability, very high performance, and greater flexibility and range of application than today's systems. While current high-performance commercial graphics systems employ various forms of parallelism to support interaction with 3D objects and scenes, all operate on a single stream of graphics primitives. Much of our effort has been devoted to finding ways to operate in parallel both on lists of graphics primitives as well as on the pixels within primitives, and to incorporate parallel execution of user application code to generate and modify the graphics primitives to be displayed.

During the past year, our team completed a full-scale prototype of Pixel-Planes 5, a machine with considerable power and generality for rendering curved surfaces, textures, and digital imagery, and modularity to allow various configurations and to upgrade components. The hardware prototype became operational in early July 1990, and is now expanded to a two-card-cage system (80 nodes on a ring network). Algorithms and applications developed over the past several years on a software "porting base" migrated to the hardware platform with relative ease. During the past six months, operating system software has been tuned and refined to the extent that we are now very close to our original goal of providing performance to the user of better than 1 million Phong-shaded triangles in a real applications. There are now many regular users developing new applications for the machine, so many that we have begun routinely reconfiguring our two-rack system, alternating configurations between a single, large, more powerful system for advanced algorithm development and two smaller systems to support more users. A third rack is under construction, which will complete the build for a full-scale machine. However, this equipment will likely be configured as separate 2-rack and 1-rack machines for much of the time, to support certain particularly intensive users, such as the VISTAnet project.

Current graphics architectures, including Pixel-Planes 5, have certain limitations that are increasingly painful for algorithm and application programmers. These include limited performance in generating image-based textures, lack of fast anti-aliasing, and long latency between user input and scene update (particularly troublesome for head-mounted displays). We are also increasingly interested in attacking the very difficult problem of real-time volume display of data from real-time ultrasound imaging; current architectures are orders of magnitude away from performance levels needed in this application. These limitations have motivated a new effort to explore multicomputer graphics architectures based on image composition. Two complementary approaches have been explored in some detail. We plan to develop one of these approaches more fully in the near term on an experimental prototype.

During the past year we effected a limited license of our patents to IVEX Corporation (Norcross, GA) for use in vehicle and flight simulation and training. One of our senior staff, Thomas ("Trey") Greer, has joined IVEX's staff, but continues to work in our laboratory developing algorithms (that remain in the public domain) for future IVEX products based on our technology.

2. RESEARCH PROGRESS

2.1 Research Issues

We begin by discussing our motivation for exploring multicomputer models for high performance graphics systems. In general, all 3D graphics rendering systems have two parts: (1) a transformation engine that computes the display-screen-space position of each bounding edge or vertex of a collection of primitive objects (*e.g.*, polygons) that make up a digital scene, and (2) a rasterization engine that determines visibility for each primitive object and that paints each primitive onto pixels on the screen. Current systems can deliver performance to the user of a few hundred thousand 3D primitives per second. At substantially higher performance levels, there is too much work at the front end for a single geometric transformation engine and at the back end for a single rasterization engine.

If we want to render polygons (or other primitives) at rates high enough that dozens or more concurrent processors are needed to provide sufficient computation power, both the transformation and rasterization tasks need to be distributed across multiple processors. If we assume that an application has the characteristic that its graphics primitives change little from frame to frame, then distributing the transformation task requires distributing the primitives of the graphics data base. The viewpoint of the final rendered scene is arbitrary and initially unknown. However, a "random" distribution of the primitives across multiple processors usually results in uniform loading of the independent transformation engines. Each of these engines will generate transformed graphics primitives that can fall anywhere on the screen image. This re-sorting onto the screen is the main problem to solve in distributing the rasterization task across multiple processors. There appear to be three distinctly different ways of solving the problem, illustrated in Figure 1.

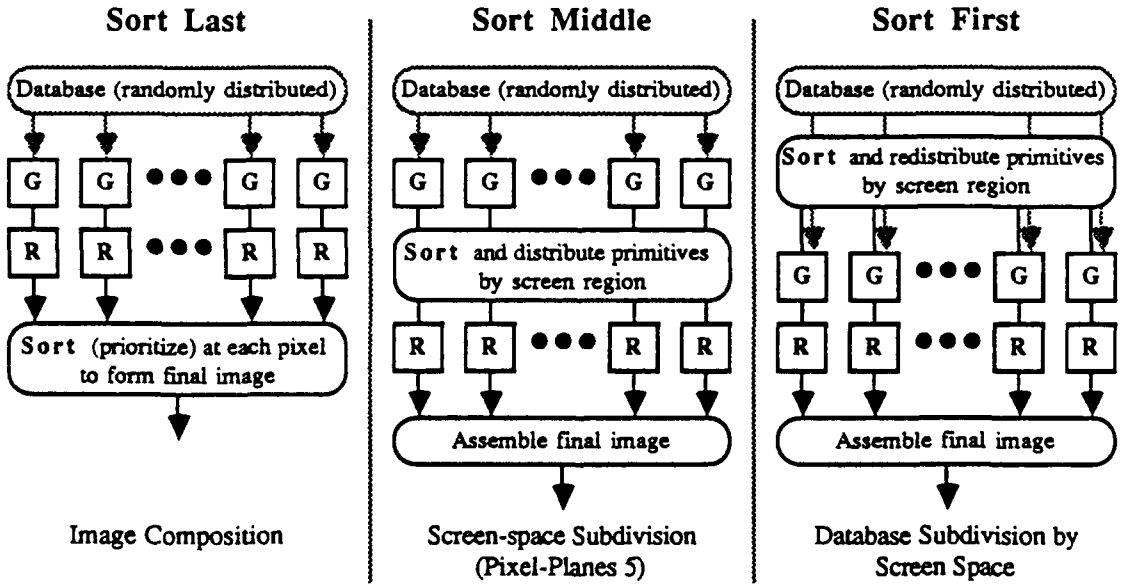

Figure 1. Three methods for distributing graphics computations over a multicomputer. Nodes labelled "G" are general-purpose, math-oriented processors for transforming primitives, "R" nodes are rasterization processors.

Sort Last (Image composition). Transformation engines and rasterizers are paired, so that a given rasterizer paints primitives only from its own transformation engine. Transformed primitives therefore require only a local path to the rasterizer. Each rasterizer paints a *partial* image of the *entire* screen. The partial images then have to be *composited*(by z-buffer or similar methods), a priority sorting that eliminates surfaces in one partial image hidden by those in another.

Sort Middle (Screen-space subdivision). Transformation and rasterization nodes are independent; primitives are sorted by screen-space subdivision and routed from transformation engines to rasterizers. Each rasterizer paints a *complete* image of a *portion* of the screen image; the final image fragments are collected, once rasterization is complete, and assembled in an image buffer for display.

Sort First (Database subdivision by screen space). Like the screen-space subdivision idea, each rasterizer paints a complete image for a portion (or portions) of the screen, and image fragments are collected and assembled in an image buffer for display. The distinguishing feature of this approach is that after the initial random distribution of primitives, primitives are re-distributed by screen-space region to the appropriate transformation engine/rasterizer pair. Assuming there is some frame-to-frame coherence, thereafter transformed primitives are moved around mainly locally.

These approaches have complementary advantages and disadvantages:

Sort last, (image composition) has the advantage that no sorting and distribution of primitives is required. The main disadvantage is that the image composition step requires a high-bandwidth "smart" network to perform the final pixel sorting (prioritization). Since the network will has to operate at high speed to maintain frame-rate image updates, special processors, tailored for this task, are required, and this restrict the kinds of rendering algorithms that can be supported.

By putting the sorting earlier in the process, both the screen-space and database subdivision approaches have the advantage that a given rasterizer has all of the information needed to determine the final color of a pixel. These approaches therefore lend themselves to a wider variety of rendering methods, although the sorting required is more complicated. Screen-space subdivision has the disadvantage that all of the transformed primitives must be sorted by screen region and routed to the appropriate rasterizer; for this purpose, some general communication network is required, and it must transfer all of the transformed primitives in the database on every frame. This is exactly the approach we have taken in Pixel-Planes 5.

The technique of earliest sorting, database subdivision by screen space, moves primitives to the transformation-engine/rasterizer pairs that handle the part of the screen on which they fall. Each time the user changes the database in some way (say, by changing the viewpoint), the system will have to perform some amount of redistribution, and the software to make this happen may be fairly complex. The advantage is that the (general) interconnection network only has to carry the traffic of primitives that move from one rasterizer region of the screen to another. Since there is inherent spatial and temporal coherency in the distribution, we speculate that there may be significant advantages in this approach, though as yet we have no experimental results.

By putting the sorting at the beginning or at the end, the transformation engine and rasterizer can be combined into a single hardware unit, with higher bandwidth between the G and R units. These nodes are each themselves complete graphics systems, so many of the architectural approaches for standard graphics pipelines can be applied. Furthermore, only a single hardware unit needs to be designed, built, and programmed.

We are currently investigating all three of these approaches. The ramifications of the sort-middle approach are being worked out in considerable detail on Pixel-Planes 5. The sort-last approach has been the focus of much effort over the past six months, and looks extremely promising. We are therefore planning to build a hardware prototype in the coming year that exploits this approach. We are also exploring implementation of a sort-first architecture on a commercial multicomputer with a custom distributed frame buffer of our own design.

2.2 Image Composition Graphics Architectures

In this section we detail some of our recent investigations into sort-last architectures.

In an image composition or sort-last architecture, a number of identical graphics engines are each assigned a subset of the primitives in the database. During rendering, each computes an image of its fraction of the primitives. The output of each graphics engine is fed into an image-composition structure that combines them into a single stream of pixels that describes the entire image (see Figure 2).

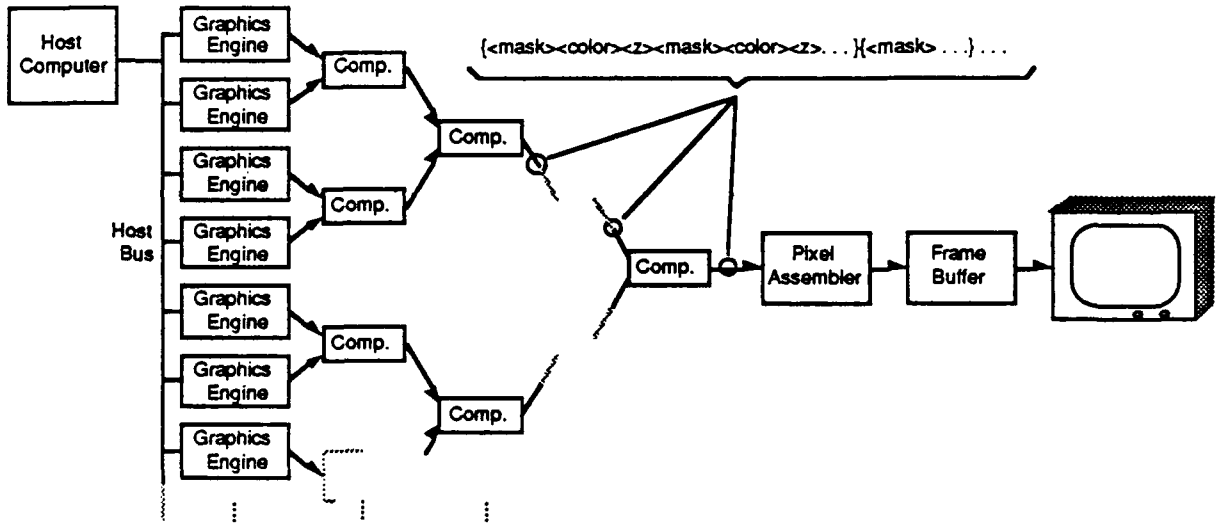

Figure 2. Conceptual block diagram of an image-composition system.

Image composition architectures offer the following potential advantages:

- **Scalability.** Since the format of an image is independent of the number and types of primitives that were used to generate it, any number of images can be composited at the same marginal cost.

- **Simple programming model.** Aside from database distribution and editing, the rendering process on each renderer is independent of all of the others. The parallelism of the overall system is hidden to the programming model for an individual renderer.

- **Low latency.** If the rendering algorithm is structured properly, the image generation latency

can be reduced to close to just over one frame time. The minimum possible latency is one frame time.

- **Flexibility.** Image composition is independent of the algorithm used to generate the subimages. Subimages could be generated by a variety of algorithms that produce images of the proper format.

There are two main approaches toward building an image composition system: z-buffer-based, and A-buffer based. The distinction is due to the rendering method used in the individual renderers and determines the properties of the image composition network and the style of anti-aliasing that can be done.

Z-buffer systems compute an image using a color and z-buffer memory array. After the image is computed, color and z values are scanned out of the array and send to the compositor. The compositor compares the z-values of corresponding pixels from the current image and from previous images, and forwards the pixel representing the closest surface.

A-buffer systems are similar, except that they compute coverage information for each surface that partially or completely covers a pixel. This information is used to build a list of potentially visible surfaces (called *fragments*) for each pixel. The composition task is more difficult in an A-buffer system: for each pixel, fragments from the current renderer must be interleaved with fragments from upstream renderers. Some fragments may be deleted because they are completely covered by fragments from the other pixel stream.

A-buffer systems handle anti-aliasing automatically: the coverage information stored with each fragment can be used to compute a final pixel color that is weighted by appropriate contributions from each visible fragment. Z-buffer systems must anti-alias by supersampling: rendering the image at higher resolution and blending the multiple samples that correspond to each pixel (in practice this is generally done by rendering the image multiple times with sub-pixel offsets, and summing the results using appropriate blend factors).

2.2.1 Design objectives

Both z-buffer and A-buffer styles of image composition are potentially useful tools for building high-performance, high-image-quality graphics systems. Image composition is particularly intriguing for us because of our group's long-term interest in highly interactive graphics systems. We decided to see if either of these methods could be used to build a system with better performance and interactivity than existing systems.

Image composition is practical only when renderers are large compared to nodes in the image composition network. We believed that a system containing 100,000 polygon-per-second renderers would reduce the cost of image composition to an acceptably small fraction of the overall system cost. In order to be useful, such a system must demonstrate high-performance, low latency, flexibility, programmability, and cost-effectiveness. In addition, we wanted the architecture to be general-purpose enough that we could experiment with algorithms and applications, as we have with previous prototype graphics systems. The following are design

objectives which we felt were essential for a research image-composition system:

- **High performance.** This was the most important design objective. Image composition potentially offers scalability to arbitrarily high performance. We wanted the prototype system to demonstrate this. In addition, to be feasible, the system should achieve this performance with a reasonable amount of hardware. A goal of 2-3 million triangles per second became a rough target for a one-rack system.

- **High frame rate and low latency.** This was a second crucial objective for real-time applications, such as flight simulators and head-mounted displays. Such applications demand frame rates of 24-30 Hz. or higher. Low latency is equally important, since lag between input sampling rate can cause confusion and even nausea for users of simulators and head-mounted displays.

- **High resolution, anti-aliased display.** High-performance graphics systems must generate high resolution, high quality images. The current standard display resolution is 1280x1024 pixels. We also wanted the system to be able to generate reasonably anti-aliased images without sacrificing the update rate or resolution. This mandates extremely high bandwidth in the image composition network.

- **Support for advanced primitives and shading models.** Applications such as computer-aided modeling systems or scientific visualization must be able to trade image quality for rendering time. We wanted the system to be flexible enough so that the same system resources could be used to render simple images rapidly, or to render sophisticated images with curved surfaces, volume data, Phong shading, multiple light sources, high-quality anti-aliasing, or texturing, as rapidly as possible.

- **Economically and technically feasible.** Our research group has limited financial and personnel resources. In order for this to be a feasible project, we had to minimize the number and complexity of components that needed to be designed, and to make use of off-the-shelf or existing parts when available. To be affordable and to minimize technical risk, we wanted to avoid exotic or expensive parts and technologies.

2.2.2 System overview

We have developed a high-level design for an image-composition system that accomplishes the objectives above. The system is composed of one or more 19-inch racks containing approximately 18 renderer boards each. The system is modular and can be configured with an arbitrary number of renderer boards. A subset of the renderers can be configured as shaders for more sophisticated rendering algorithms such as Phong shading, volume rendering, and anti-aliasing. A one-rack system will be capable of rendering approximately 800,000 z-buffered, Phong-shaded, 100-pixel triangles per second. The system can be extended with additional racks for proportionately higher performance.

A one-rack system contains three main board types:

- *Renderer/Shaders*, which are one-board graphics computers capable of rendering 100,000 z-buffered triangles per second or computing shading models for 160x128 pixels in parallel.
- A *host interface*, which implements the connection to the host computer and contains synchronization and diagnostic hardware for the image composition network
- A *frame buffer*, which buffers and displays composited pixels.

A multi-rack system contains two additional board types for extending the system backplane between adjacent racks:

- *Out* boards, for bringing backplane connections to external connectors at the front of the card cage
- *In* boards, for connecting additional backplanes to the outee board of the previous rack.

Figure 3 shows a logical diagram of a one-rack system.

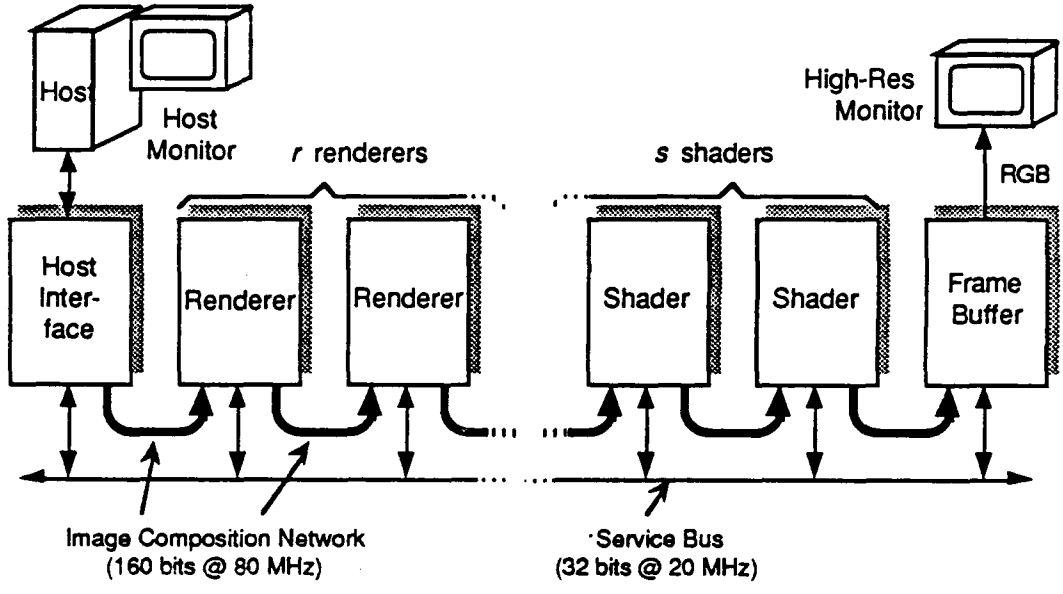

Fig. 3: Logical diagram of the image composition system.

All logical boards in the system plug into a common backplane, which contains the service bus, the image-composition network, and wiring for clock and power distribution.

The service bus is a general-purpose bus controlled by the DECStation 5000 host computer that allows the host to randomly read and write data to any board in the system. It is used for loading code and data during system initialization and for updating viewing parameters and editing the graphics database during interactive operation.

The image-composition network is a very wide (160-bit), high-speed (80 MHz) special-purpose communication network for rapidly moving pixel data between adjacent boards. It performs two

functions in different parts of the system: it transfers pixel data between compositors in the renderer boards and it transfers rendered pixels from the shaders to the frame buffer. The image-composition network is implemented as 160 wires that connect adjacent backplane slots. Compositor chips on the renderer and shader boards synchronously transmit data unidirectionally to compositors on adjacent boards.

The following sections describe the architectural issues and decisions that were made to arrive at the final system design.

Z-buffer vs. scan-line rendering

Initially, we believed that *z*-buffer rendering combined with supersampling required too much bandwidth to be practical in a moderate-cost system. Two data words (color and *z*) are required for each sub-pixel sample, and conventional wisdom says that 8 to 16 samples are needed for adequate sampling. As a result, we focused our early efforts on scan-line-based A-buffer systems. Two factors caused us to revisit z-buffer based systems:

1) The realization that, to avoid serious artifacts, an A-buffer algorithm requires four data words per fragment and two to three fragments per pixel in a complex image—a total of 8–10 words per pixel.
2) The development of techniques to reduce the number of samples per pixel (see Molnar's paper on efficient anti-aliasing, referenced below).

Since supersampling requires only two words per sample, five samples per pixel require no more bandwidth than the two to three fragments per pixel in an A-buffer system. By choosing sample locations and weights carefully, we found that five samples per pixel are sufficient to anti-alias most scenes for most applications. Supersampling also avoids artifacts that arise from sampling the color only once per pixel. Furthermore, the compositor in a *z*-buffer system is simpler, implying that it can be smaller and run faster than an A-buffer compositor. With this renewed interest in *z*-buffer systems, we began looking for ways to build high-performance, compact *z*-buffer renderers.

Renderer design

One of the current design efforts in our laboratory is the development of a one-board graphics engine for polygon rendering within a IVEX Corporation's flight simulator (see Section 2.4). Trey Greer determined that a single Pixel-Planes 5 Graphics Processor combined with a Pixel-Planes 5 Renderer can render 50,000 anti-aliased triangles per second and 100,000 triangles per second without anti-aliasing.

A one-board renderer of this type satisfied our need for high rendering performance in a compact package. Since it had much in common with Pixel-Planes 5, it would be straightforward to design. The serial ports of the EMC chips led to a natural, high-bandwidth interface to the image composition network: a renderer containing 64–80 EMC chips has 256–320 serial communication wires that can transmit pixel data at 20 MHz.

Image composition network

The compositor in each renderer board has to merge the 320 40 MHz wires from each renderer with the pixel stream from upstream renderers. We had to find an appropriate implementation for the image composition network. The two main choices were:

- *Fast and narrow.* We could multiplex the signals up to use fewer, high-speed wires. This would require level conversion to a higher speed technology such as ECL as well as multiplexing circuitry.
- *Wide and slow.* We could leave the network wide and use conventional TTL or CMOS technology to do the compositing and transmit pixel data between boards, though many wires and connectors would be required.

Initially, we considered the fast and narrow approach. ECL circuitry can run at speeds up to several hundred MHz with relatively straightforward design methods and little risk. This minimizes the number of wires connecting boards. Although we have had positive experiences with the 160 MHz 32-bit wide ring network in Pixel-Planes 5, it requires disciplined design and the problems scale as clock rates increase. We would also need to multiplex up the parallel data on each board. This would be very expensive in terms of parts and board area, particularly considering the cost of TTL/ECL level conversion.

The technical problems seemed more difficult with a very fast network, and the savings in parts was not substantial. The format of pixels transmitted by an EMC led to a simplification that tipped the balance in favor of a wide network: each EMC chip has two 2-bit ports, over which pixels are written in two-bit-serial fashion. Compositing requires comparing z-values of corresponding pixels. Since two-bit comparisons are much simpler than word-parallel comparisons, the serial nature of the EMC serial port led to the notion of serializing the composition calculations, allowing them to be performed in simple, fast programmable logic devices.

The compositor can be implemented in a 22VP10 PLD. Gazelle manufacturers reasonably priced 22VP10's with 6 ns clock-to-output propagation delays and 2 ns setup times. The compositor PLD's can be positioned near the board edge, minimizing the wire length between parts. All of these factors meant that we could design the image composition network to run at 80 MHz with 160 wires passing from board to board, a number readily supportable using vanilla connector technology.

Support for deferred shading ("end-of-frame" calculations)

One of the most successful features of Pixel-Planes 5 is its ability to factor out, and perform in parallel, calculations that need to be done only once per pixel. We call these end-of-frame calculations. They are particularly useful for shading calculations, which are time-consuming, but are primitive-independent; they only depend on generic quantities such as surface color and surface normal vectors.

Pixel-Planes 5 performs these calculations efficiently by waiting until all the primitives in a region have been rendered and then performing end-of-frame calculations for the region. The image-

composition system, on the other hand, is at a major disadvantage compared to Pixel-Planes 5. In a naive implementation, end-of-frame calculations would have to be performed once for each region for each renderer—increasing their expense by an order-of-magnitude or more.

We realized that Pixel-Planes 5 EMCs are ideal processors for end-of-frame calculations. The problem is that the data is in the wrong place at the wrong time. A solution to the problem is to designate a subset of the renderer boards for use as shaders. Rendered pixel data from the renderers would then be composited in the normal way, but fed into a shader, rather than proceeding directly to the frame buffer. If pixel attributes such as intrinsic color, surface normal vector, and z value are sent down the image-composition network instead of RGB values, the shader can perform end-of-frame calculations, forwarding final rendered pixel values when it is finished shading. (The PLD's in the renderers can be configured to allow this type of operation as well as the normal compositing process).

This enhancement allows the image composition system to compute Phong and other sophisticated shading models with the same cost savings as Pixel-Planes 5.

2.2.3 Status

We have spent approximately a month arriving at a "blackboard" design for this system and in performing hand analysis of its behavior. We are sufficiently encouraged that we now plan to undertake a full, detailed design of the hardware. Concurrently we will build a software simulator for the system capable of instrumentation to gather data about performance under overload and load imbalance. We have also begun the construction of system and application software, based on Trey Greer's bounded system under development for IVEX.

2.3 Pixel-Planes 5 Developments

2.3.1 NTSC-Compatible Frame Buffer

Since our last report, a new Pixel-Planes 5 application board has been designed and tested. The NTSC Frame Buffer Board stores the pixel data generated by the Renderer(s), and generates RGB signals with RS-170-compatible timing for two 640x525-pixel (~640x480 pixels displayed) 30 Hz interlaced displays. The board.can be "gen-locked" to an external RS-170 level composite video stream (*e.g.*, from a camera or video tape recorder), or to a composite sync pulse stream from a studio sync generator (0 to -4 volts), or it can generate its own sync timing.

The NTSC Frame Buffer Board consists of two logical frame buffers, each with its own color lookup tables and cursor chips. The two frame buffers share control registers, pixel clock circuitry, and genlock circuitry. Each frame buffer has its own triple RAMDAC and two user-definable hardware cursor chips. The memory associated with each frame buffer is physically organized as four banks of 512 x 512 x 24-bit pixels. Logically, each frame buffer consists of 40 128x128 pixel regions. There are two identical halves of each frame buffer, each occupying 256 rows of memory. Thus, double-buffered operation is possible for each frame buffer. Switching buffers takes place only during vertical retrace so as to prevent the image from tearing. The NTSC Frame Buffer accepts packets consisting of pixel data containing a scanline or multiple scanlines

within a 128x128-pixel renderer region. A packet containing multiple scanlines may contain pixel data for only one field or pixel data for both fields.

The NTSC Frame Buffer Board will be used in applications that require the ability to synchronize the video stream to a "standard" timing source. It will be used to generate images that can be videotaped; the video camera and the board will share a common sync source.

These boards will also be used by the head-mounted display research group in our department. Since current head-mounted displays (HMD's) must rely on commercial pocket television display technology, only NTSC-quality video is needed. In addition to "conventional" HMD's, the group plans to use Pixel-Planes 5's multiple frame buffer capability in two ways: (1) Multiple HMD users in a shared virtual world, each receiving stereo images from a pair of NTSC Frame Buffers, and (2) four or more independent large conventional displays for panoramic field of view, with (field-sequential) stereo imaging and head tracking.

2.3.2 Multi-Rack, Multi-Machine Reconfiguration

We have implemented a procedure whereby multiple configurations of the Pixel-Planes 5 system are available. The full configuration of the machine is currently a two rack system; this machine is used to experiment with algorithms and applications for maximum performance rendering. However, it is also useful to have more than one instance of Pixel-Planes 5 available to increase software development efficiency (one of the limiting factors in Pixel-Planes 5 software development is contention for machine access). Consequently, we have implemented a procedure whereby the machine is reconfigured on a regularly weekly cycle: from Thursday noon through Monday morning the machine is configured as the maximal 2-rack system, and during the remainder of the week it is configured as two lower performance 1-rack systems.

The ability to do these reconfigurations is afforded by the high degree of modularity in the Pixel-Planes 5 architecture. First, it is possible to configure the communication Ring to span one or more card cages; secondly, the architecture is scalable so that varying numbers of GP (i860) and Renderer (parallel SIMD pixel-processor) boards can be installed in a system; thirdly, the system software can adapt to the machine configuration at run-time so the software need not be re-compiled for various machine configurations. The reconfiguration is accomplished is as shown in Figure 4. Each card cage contains a balanced complement of GP and Renderer boards, plus a Host Interface board, a hi-res or low-res (NTSC) Frame Buffer board, an Inee board, and an Outee board. To configure a 1-rack system, a pair of ribbon cables connect the Inee and Outee boards to complete the Ring loop. To configure a 2-rack system, the ribbon cables are cross-coupled (so to speak) so that the Inee board in one card cage connects to the Outee board in the other card cage; thus one Ring structure is spread across 2 card cages and backplanes. A toggle switch on the front panel of one Host Interface board is switched to disable that board, so that the Ring has a single Host Interface. Thus, reconfiguring the system can be accomplished in minutes by switching the ribbon cables around, renaming on-line configuration files describing the hardware configuration to the software, flipping a switch to enable or disable the second Host Interface board, and posting a wall message on the host machine. (It is even possible to install two Host Interfaces and Inee/Outee pairs in the same card cage so that two systems can reside in one card cage.)

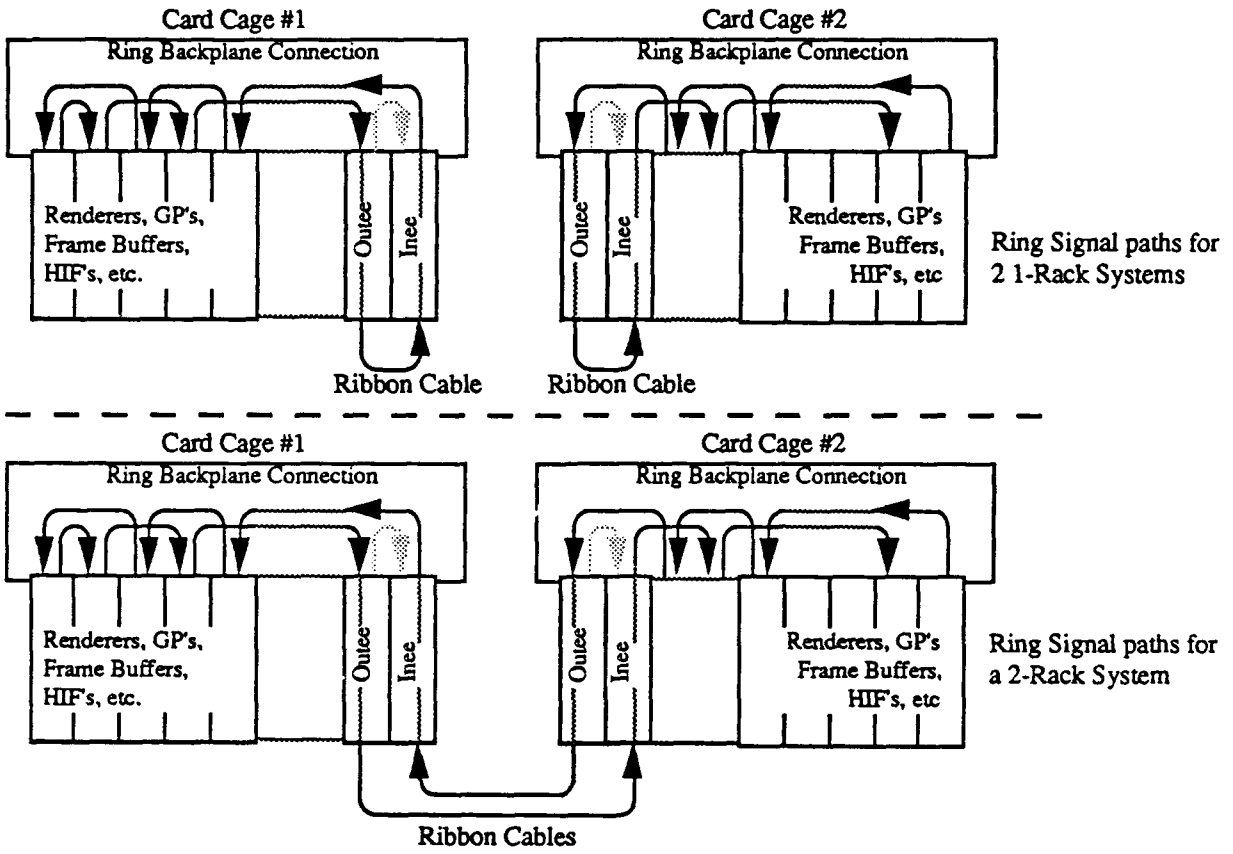

Figure 4. Cabling options for 2 1-Rack systems and for a single 2-Rack system.

We are planning to build a third rack; when this occurs we will be able to configure three 1-rack systems, a 1-rack plus a 2-rack system, or one big 3-rack system.

2.3.3 IGC Microcode

On the Pixel-Planes 5 Renderer board, the bit-serial SIMD computations in the array of pixel processors are managed by a custom VLSI chip known as the Image Generation Controller (IGC). This part contains a microcode sequencer to generate the micro-instructions for the pixel processors and a set of shifters for bit-serializing the quadratic expression evaluator coefficients. The instruction set for the IGC contains commands for performing the steps necessary to render various primitives (i.e.: a command to process an edge equation, a command to perform the Z-buffer comparison step, and a command to linearly interpolate shading information).

We have long believed that this SIMD array of processors might be useful for performing calculations other than pixel-oriented graphics operations, such as scientific calculations that require evaluating polynomial functions over a regular grid of values. To encourage such use of the Renderer, we are trying to extend the instruction set of the IGC so that it constitutes a fairly complete set of operations, not just those useful for pixel-oriented rendering calculations. The instruction set for the IGC has evolved to look very similar to the instruction set for a simple microprocessor, but with two differences. First, the variables, instead of having fixed lengths like 8, 16, or 32 bits, can be arbitrary in length because of the bit-serial nature of the SIMD processors;

this yields efficiencies both for memory allocation and for computational time (i.e.: if only 11-bit values are needed for some group of variables, computations take only 11/16 as long a computations on 16-bit quantities). Second, there is a special variable, the local value from the quadratic expression evaluator, which can be used in arithmetic and logical operations just like variables from the local memory.

One drawback to this expansion of the IGC instruction set is that we are running out of microcode memory in the IGC chip, so we will probably have to implement tools that allow an application to compile custom IGC microcode that includes only those instructions needed by the application.

2.4 Technology Transfer to IVEX Corporation

Over the past year we have licensed our patents to IVEX Corporation of Norcross, Georgia, a maker of flight and vehicle training visual systems. Recently the pace of joint work has increased substantially. A long-time staff member on our team, Trey Greer, has become an IVEX employee, charged with developing a polygon generator board product, based on the Pixel-Planes 5 Graphics Processor and Renderer hardware. The polygon generator board will supply anti-aliased polygons representing aircraft, ground structures, and other scene elements at frame rates, to be overlaid on textured terrain and light effects supplied by other IVEX hardware.

Through a short-term contractual arrangement between IVEX and UNC, Greer maintains an office in our laboratory and access to the Pixel-Planes 5 prototype for algorithm development. He has concentrated on algorithms for rendering anti-aliased images using a small system consisting of a single GP and a single Renderer. Current performance figures for this small system generating images on a 640 x 512 pixel screen are: 1,711 triangles at 30 Hz update rates with 5-sample anti-aliasing and 1,290 triangles at 17 Hz update rates with 15-sample anti-aliasing, both with Gouraud shading and haze effects.

The work Greer is performing for IVEX has had significant benefits for our own research. The effort to squeeze maximum performance out of a minimal system has led to several ideas for increasing performance on the much more general software of Pixel-Planes 5. These include two ways to reduce the work of the Graphics processor in delivering transformed primitives to Renderers:

° IGC command templates. Build templates of primitive rendering commands for the Image Generation Controller (IGC) on a Renderer, then simply plugging primitive data into the template at run time.

° Indirect write queue and polling. Place pointers to the templates in a queue for transmission to the Renderer, avoiding memory-to-memory copies. The write queue is emptied onto the Ring network via a polling paradigm, avoiding the context switching overhead of the interrupt-driven schemes we have been using.

The tight focus of Greer's work has proved an important motivation for the design effort on the image composition prototype we have described above, and the code for the IVEX polygon generator has become the starting point for developing software for the image-composition hardware node. Greer's work has also been a test bed for evaluating some recent research results on generating efficient anti-aliasing sampling kernels, and has incidentally provided us with a very elegant but simple tool for adjusting the gamma correction on our laboratory's video monitors.

2.5 Graphics Algorithms and Applications

System Software Overview

Pixel-Planes 5 system software has three layers: the Ring Operating System (ROS), the Rendering Control System, and applications software, including the PHIGS+ compatible graphics library (PPHIGS).

The Ring Operating System (ROS) is the basic operating system for Pixel Planes 5. It provides for asynchronous sending and receiving of messages via the ring network among all the application boards, and provides automatic execution of user-defined event handlers.

The Rendering Control System (RCS) performs coordination of the Graphics Processors, Renderers, host, and Frame Buffers. Basically it is an executive program for graphics applications that runs on top of ROS. It makes calls to the PPHIGS library routines at the appropriate times during the basic frame rendering loop. The frame rendering loop is pipelined so that at any moment two frames are in progress, to achieve better utilization of the hardware. The Rendering Control System is directed via messages from an application program running on the Pixel Planes host. Via library calls, the application program updates the graphics data base or calls for generation of a new frame. If a frame is completed before the application program calls for a new frame, refinement frames are automatically generated to produce an improved, anti-aliased image.

Graphics Applications

Work on PPHIGS continues. Several features implemented over the past several years on Pixel-Planes 5's software porting base are now debugged and running on the hardware. These include anti-aliasing by adaptive refinement using supersampling and multi-pass transparency. The multi-pass transparency is employed by one user to help visualize four-dimensional shapes. We have also added the capability to use more than one frame buffer at once to show multiple views, including stereo pairs.

Three volume rendering applications are under development. The first, called *vrn*, produces 3D renderings of volume data sets, such as CAT scan data at rates of about 2 frames per second for a 128^3 data set. The purpose of this program is to provide a tool for investigating medical imaging techniques.

Many features have been added to *vrn* since our last report. An interactively controllable cut plane can be used to make the volume on one side of the plane transparent. Fairly intuitive joystick controls allow the user to orient and position the cut plane, providing the ability to cut open the volume and see inside it. This capability is particularly helpful with medical MRI data, which has a large amount of detail internal to the volume. The cut plane can be highlighted by making it partially opaque; this feature improves understanding of the relationship of the cut plane to a complex data set. In addition, the user can add a wire-frame bounding box around the volume data to provide reference points.

vrn now scales volumes non-uniformly during rendering. This feature allows non-cubic voxels to be stretched until the volume appears to have the correct aspect ratio. Before this feature was implemented, the program created copies of certain volumes that needed to be stretched along an

axis, so the new feature greatly improves disc usage and run-time performance.

A second volume renderer, developed by Tim Cullip of the UNC Department of Radiation Oncology, operates much more rapidly than *vrn* producing 10 frames per second. Its major design criterion was speed at the expense of flexibility and data set size. In order to achieve the speed, it replicates the entire data set on every GP node and assigns a contiguous screen region to each GP (the Renderers are not involved in the calculation at all, so this approach would work well on other multicomputers, assuming a fast distributed frame buffer). One major capability of Cullip's renderer, "synthetic reprojection", has now been incorporated into *vrn*. Synthetic reprojection simulates a transmission xray by combining samples along a ray in various new ways.

A third volume renderer is under development, based on Lee Westover's "splatting" technique, which uses forward projection from volume elements onto the screen, rather than ray tracing. This application makes extensive use of the Renderer's ability to evaluate in parallel quadratic expressions on screen space, in order to compute point spread function filter kernels for each volume data point. Currently written in C (no microcode), this volume rendering program generates images at rates of 1-5 frames per second for 128^3 data sets. Future work will focus on enhancing speed through optimizations and progressive refinement. Additionally, a new approach for merging polygons with volume data will be explored.

The building walkthrough project is currently working with a 40,000 polygon database of a complete house, illuminated using a Radiosity model, and making extensive use of some 20 procedural textures. While the model and user interface have improved, update rates are still only about 2 frames per second.

This low update rate is mainly attributed to the computational expense in the GP's of generating texturing macros, long streams of Renderer instructions needed to produce the procedural textures. Work is progressing to speed up texturing and shading considerably by taking advantage of the fact that texturing macros and other shading operations rarely change between frames. They may therefore be pre-computed and re-used.

The texture generation language has been enhanced by adding condition codes that can enable and disable individual pixels in a texture. Users have found that this facility significantly increases the flexibility of the texture codes, and may also allow texture optimization by combining several similar textures into a single instruction sequence only slightly larger than the individual sequences.

Work is progressing on direct rendering of arbitrary second order (quadric) surfaces on the Renderers. Currently the system supports fast rendering of spheres, but the spheres do not transform properly under arbitrary rotation, translation, and scaling. The new rendering algorithm will be slower, but will properly transform objects, allow new primitives, such as cones and cylinders, and will support texture mapping.

Performance Improvements under PPHIGS

We have made considerable progress in speeding up support for graphics applications under PPHIGS, and most of this effort has been expended on optimizing the inner loop for triangle rendering. All the routines involved in this inner loop, including traversal of the display list, transforming triangles, copying bins, and so forth, were optimized both in terms of time and space. All of the relevant routines were then merged into one code block and further optimized.

The whole block now fits into the i860's 4 Kbyte instruction cache, which is critical for achieving optimal performance from the i860. The current code yields performance of 660,000 Phong-shaded triangles per second on a one-rack system with 8 GP's (16 i860's) and 8 Renderers, with images on a 1280x1024 screen; we expect shortly to achieve performance well in excess of a million Phong-shaded triangles per second in a two-rack system. Approximately 60% of the GP's time is spent in the triangle code block; the rest is mainly message passing overhead, which will be optimized over the next few months.

Faster Polygon Rendering Algorithm

We have recently begun investigating algorithms for more rapid polygon generation on Pixel-Planes 5, for applications that do not need some of the advanced features currently implemented on our original PPHIGS-based renderer (*e.g.*, textures, transparency). Since a simple rendering does not need many bits per pixel, in this algorithm we allocate multiple regions to each Renderer, typically four, all of which are held on-chip in our logic-enhanced memory devices (giving 208/4 bits for each pixel on the screen), in which we RGB and z. With 16 Renderers, for example, each with 4 x 128x128 pixels, we can cover a 1024x1024 window on the screen. The data base of triangles is divided among all the GP's. Each GP, as it transforms and clips, performs lighting calculations on a triangle, then immediately sends it off to the appropriate region in the appropriate Renderer. Contention of multiple GP's for the same Renderer is handled by the Ring network's hardware protocols. Preliminary experiments indicate that this approach may yield dramatically higher performance than our original, rather more general purpose rendering algorithm.

2.6 Progress in the VISTAnet Project

(contributed by Jim Symon, UNC Department of Radiation Oncology)

The current VISTAnet system connects a medical workstation (MWS) in the UNC Department of Radiation Oncology to the Cray Y-MP at the North Carolina Supercomputing Center (NCSC) and to the Pixel-Planes 5 graphics engine at the UNC Department of Computer Science. Initially all inter-process communication (IPC) was made through the existing Internet connections. We now have a dedicated T3 fiber link (45 Mbits/sec) connecting the Cray with Pixel-Planes 5. This represents an intermediate stage before the OC12C link (622 Mbits/sec) is completed next winter.

In an XWindow-based user interface on the MWS, a researcher interactively manipulates simple 3D wireframe representations of patients and radiation treatment beams. With each change, the MWS transmits new beam information to the Cray. The Cray performs the compute-intensive dose calculations establishing the radiation dose level at every point in the patient anatomy. The Cray process sends the resulting dose data to Pixel-Planes 5, which performs various rendering tasks including synthetic reprojection of CT scan data with the dose distribution embedded. Pixel-Planes 5 displays the rendered images on a monitor adjacent to the MWS.

Until now, standard IPC socket mechanisms transferred data and control information. The T3 link allows direct injection of data from the Cray into the Pixel-Planes 5 Ring and its individual components. The Cray behaves like a Pixel-Planes 5 host despite being located some 20 kilometers away. With the OC12C link VISTAnet should achieve its goal of real-time, interactive calculation and display of radiation treatment dose information within high quality rendered images. The power of the Pixel-Planes 5 graphics engine will play an essential role in reaching this goal.

3. RECENT PUBLICATIONS

Molnar, S., "Efficient Supersampling Antialiasing for High-Performance Architectures," (submitted for publication).

Neumann, U., Yoo, T., Fuchs, H., Pizer, S., Cullip, T., Rhoades, J., and Whitaker, R., "Achieving Direct Volume Visualization with Interactive Semantic Region Selection," UNC-CS Tech Report TR91-012, January 1991 (submitted for publication).

Neumann, U., "Taxonomy and Algorithms for Volume Rendering on Multicomputers," UNC-CS Tech Report TR91-015, January 1991 (submitted for publication).

Poulton, J., "Building Microelectronic Systems in a University Environment," invited presentation, paper to appear in *Proceedings of the 1991 Conference on Advanced Research in VLSI*, UC-Santa Cruz, March 25-27, 1991.